Interactive Manipulation of Rigid Body Simulations

Jovan Popović[*] Steven M. Seitz Michael Erdmann Zoran Popović[†] Andrew Witkin[‡]

Carnegie Mellon University
[†]University of Washington
[‡]Pixar Animation Studios

Abstract

Physical simulation of dynamic objects has become commonplace in computer graphics because it produces highly realistic animations. In this paradigm the animator provides few physical parameters such as the objects' initial positions and velocities, and the simulator automatically generates realistic motions. The resulting motion, however, is difficult to control because even a small adjustment of the input parameters can drastically affect the subsequent motion. Furthermore, the animator often wishes to change the end-result of the motion instead of the initial physical parameters.

We describe a novel interactive technique for intuitive manipulation of rigid multi-body simulations. Using our system, the animator can select bodies at any time and simply drag them to desired locations. In response, the system computes the required physical parameters and simulates the resulting motion. Surface characteristics such as normals and elasticity coefficients can also be automatically adjusted to provide a greater range of feasible motions, if the animator so desires. Because the entire simulation editing process runs at interactive speeds, the animator can rapidly design complex physical animations that would be difficult to achieve with existing rigid body simulators.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; G.1.7 [Numerical Analysis]: Ordinary Differential Equations—Boundary value problems

Keywords: Physically Based Animation, Animation with Constraints

1 Introduction

Physical simulation programs provide powerful tools for creating realistic motion in animated shorts and feature films. These methods enable quick and easy generation of complex physical behaviors such as a ball bouncing, window breaking [22], cloth folding [2], and water flowing [10, 24]. An attractive feature of physical simulation is that the animation is generated automatically—the animator only needs to specify a few physical parameters such as initial positions and velocities.

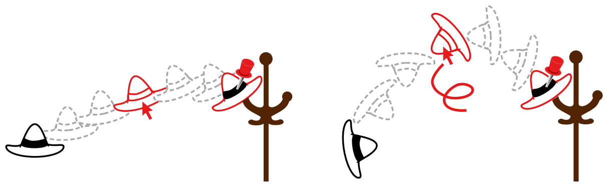


Figure 1: The animator manipulates the simulation by first fixing the hat's landing position on the coatrack with a "nail" constraint. While the animator rotates the hat at an earlier time to achieve the desired spin, the constraint maintains the desired landing location.

Despite the appeal of simulation techniques, their primary drawback is lack of intuitive control over the resulting motion. The animator often wishes to adjust the motion to achieve a desired effect such as a new end-position or a more pleasing look. However, directly altering the underlying physical parameters to try to achieve the desired effect is often cumbersome and nonintuitive. In many cases, the animator would prefer to edit the animation itself, by direct manipulation of positions and velocities.

We introduce a novel interactive technique for manipulating rigid body simulations. Throughout the interaction, our system displays the entire trajectory of all objects in the scene. The animator is free to manipulate the *entire* motion by grabbing and changing the state of the object (position, velocity, etc.) at any location on its trajectory. For example, suppose the animator wants to design a scene in which an actor successfully tosses his hat onto a nearby coatrack, but instead has an animation of the hat falling to the floor. In our paradigm, the animator first selects the hat at its landing position and simply drags it onto the coatrack. There are many ways in which the hat can land on the coatrack and the current motion may not have the desired style. The animator can adjust the style by first fixing the landing position on the coatrack to ensure the desired landing location and then rotating the hat at an earlier time until the hat motion achieves the desired spin (Figure 1).

This hat example illustrates the use of position constraints to control rigid body animations. More generally, our system provides the ability to set arbitrary position, orientation, and velocity constraints on multiple objects at multiple points in time. Furthermore, we also provide floating time constraints that may be satisfied at any point in time. For example, an animator can adjust where a mug hits the ground on its third bounce, without fixing the time when that bounce occurs.

A key problem in controlling dynamic simulations is obtaining sufficient degrees of freedom to avoid over-constrained problems. In our system the animator may add degrees of freedom by varying physical parameters that specify the internal properties of the environment, including shapes, masses, surface normals, elasticity coefficients, and moments of inertia. Often the best choice for these parameters is not at all obvious to the animator but yet can

[*]Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213–3891. `http://www.cs.cmu.edu/~jovan`

have a very dramatic effect on the resulting animation. Our system automatically adjusts these physical parameters according to the animator's desired effect. We have found this capability to be useful even in under-constrained problems. In particular, motion in chaotic systems is highly sensitive to small perturbations in the initial conditions. Adding control variables near desired manipulation times (i.e. variation of surface normals at the previous collision) improves the conditioning without affecting the perceived realism of the animation [4]. Furthermore, the additional parameters increase the range of feasible motions, enhancing the animator's ability to create interesting effects.

Interaction is an integral component of our approach. The animator is able to directly control the motion without manipulating the underlying physical parameters, and immediately sees the results of her adjustments. As a result, she can quickly explore the space of possible motions to achieve the desired behavior. Unlike previous motion-construction tools [29, 8, 21, 19, 23], our system does not evaluate the quality of motion with an objective criterion such as time-optimal motion. Instead, the animator imparts her aesthetic or other subjective criteria and interactively guides the system towards the desired motion.

Internally, our system represents the entire motion of bodies by the physical parameters that control the simulation (i.e., initial positions and velocities, surface normal variations and other parameters included by the animator). As the animator interactively manipulates the motion, the system computes the new physical parameters that achieve the desired motion update. This is achieved in real time using a fast differential update procedure in concert with a rigid body simulator. Motion discontinuities pose an additional challenge (e.g. when a point of collision changes to a different facet on a body's polyhedral mesh) because the motion changes abruptly. When this happens, our system performs a local discrete search in physical parameter space to compute the motion that most closely complies with the desired adjustments.

The remainder of the paper is divided into six sections. In Section 2, we discuss related work. We outline the basic algorithm in Section 3, and discuss further details in Section 4 and Section 5. In Section 6, we outline the specifics of our prototype implementation and report on the experimental results, and in Section 7, we conclude and describe directions for future work.

2 Related Work

Dynamics and motion of mechanical systems are important to many fields. Optimal control theory provides the groundwork for maximizing the performance of evolving dynamic systems [25]. In robot path planning, kinodynamic planning refers to the problem of computing robot paths that satisfy both dynamic and kinematic constraints [9]. In computer graphics, the spacetime constraints technique for animation of characters computes optimal motion subject to constraints [29, 8, 21, 19, 23]. Other techniques [3, 6, 16] also rely on gradient information to compute the motion that satisfies certain constraints. All of the above techniques solve for the actuating forces which produce a motion. Because of this, they do not directly apply to our problem of controlling the rigid body simulations because we wish to control passive objects (i.e. objects without any self-propelling forces).

Several researchers have addressed the inverse problem of constructing a dynamic rigid body motion given the desired body configurations. Tang et al. [27] formulated this inverse problem as an optimization task and extended the genetic algorithm technique to compute solutions for a class of 2-D N-body problems. For a 2-D billiards simulation, Barzel and colleagues [4] computed successful shots using a backward search from the desired final locations of billiard balls. Chenney et al. [7] applied the Markov Chain Monte Carlo (MCMC) method to construct 3-D motions that achieve desired body configurations. The MCMC technique excels at constructing motions for systems with chaotic behavior such as the motion of pins after collision with a bowling ball. The main drawback of these approaches is lack of interactivity: these systems may require several hours to construct a solution. If the animator does not like the resulting motion, she must adjust the desired body configurations and start again. We argue that interactivity is essential when aesthetics is a primary concern.

Our interactive technique is related to the method for geometric modeling described by Witkin et al. [28]. Similar techniques have also been devised for drawing applications [13], interactive camera control [14] and others. In its treatment of motion discontinuities, our approach most closely resembles that of Harada et al. [17], which combines continuous and discrete optimization for applications in architectural design. In this approach, when the imposed architectural constraints can no longer be enforced with the continuous parameters, the solver performs a local discrete search to find a new room arrangement in which the constraints are satisfied.

3 Interactive Manipulation

Our algorithm computes the required physical parameters so that the resulting motion satisfies desired constraints. In this section, we define some basic concepts and give a top-level description of our algorithm.

3.1 Simulation Function

Following the Lagrangian approach, we describe mechanical systems in terms of their generalized coordinates and velocities [26]. A system of one or more rigid bodies is described by a *generalized state* vector $\mathbf{q}$ whose components are the generalized coordinates and velocities of the bodies in the system. The behavior of a system is described by a set of ordinary second order differential equations [26], which we write in vector form as a coupled first order differential equation,

$$\frac{d}{dt}\mathbf{q}(t) = \mathbf{F}(t, \mathbf{q}(t)), \tag{1}$$

where $\mathbf{F}(t, \mathbf{q}(t))$ is derived from the Newton's law (e.g. see Equation 8). As mentioned in Section 1, our technique varies several physical parameters—in addition to the initial position and velocity $\mathbf{q}_0$—to modify the simulation. We encode all of these parameters in the *control vector* $\mathbf{u}$, and extend the differential equation appropriately:

$$\frac{d}{dt}\mathbf{q}(t) = \mathbf{F}(t, \mathbf{q}(t), \mathbf{u}). \tag{2}$$

This equation of motion completely describes the system in free flight (i.e. when there are no collisions): integrating Equation 2 yields

$$\mathbf{q}(t) = \mathbf{q}_0(\mathbf{u}) + \int_{t_0}^{t} \mathbf{F}(t, \mathbf{q}(t), \mathbf{u})\, dt. \tag{3}$$

Collisions can be handled in a number of ways, but for computer animations the simple Poisson collision model suffices [20]. This model can represent elastic and inelastic impacts by applying instantaneous impulses to the colliding bodies. The system simulates the motion during free flight by numerically solving Equation 2. At collision times additional impulses are applied to the system. Because the control vector $\mathbf{u}$ includes physical parameters such as surface normals at collisions and elasticity coefficients, the impulse $\mathbf{I}(\mathbf{q}^-, \mathbf{u})$ directly depends on the control vector $\mathbf{u}$. At collisions the simulator maps the generalized state an instant before the collision $\mathbf{q}^-$ into the state an instant after the collision $\mathbf{q}^+$ (e.g. see Equation 10):

$$\mathbf{q}^+ = \mathbf{q}^- + \mathbf{I}(\mathbf{q}^-, \mathbf{u}). \tag{4}$$

More abstractly, given the control vector $\mathbf{u}$ the rigid body simulator computes the *simulation function* $\mathcal{S}$, which specifies the state of the bodies in the world at every point in time:

$$\mathbf{q}(t) = \mathcal{S}(t, \mathbf{u}). \tag{5}$$

In principle, the animator could manipulate the motion $\mathbf{q}(t)$ by adjusting the control vector $\mathbf{u}$. However, such a form of control would be tedious because the relation between $\mathbf{u}$ and $\mathbf{q}(t)$ is complex and nonintuitive. Instead, we would like to allow the animator to specify the state of bodies $\mathbf{q}(t_i)$ at specific times $t_i = t_0, \ldots, t_n$, and let the algorithm compute the control vector $\mathbf{u}$ that produces the desired motion. This is a difficult problem [27, 7] for three reasons. First, the domain of the simulation function $\mathcal{S}$ is high-dimensional: for a *single* 3-D body, the components of the generalized state $\mathbf{q}$ are the body's position, orientation, linear, and angular velocity (i.e. $\mathbf{q} \in \mathbf{R}^3 \times \mathrm{SO}(3) \times \mathbf{R}^3 \times \mathbf{R}^3$). Second the simulation function is highly nonlinear. A consequence of the integral nature of the simulation function is that small changes in the initial conditions can result in drastic modifications of the motion. Third, the simulation function is not continuous. Each collision event (e.g., different vertices of an object colliding with the ground) bifurcates the simulation function.

We adopt a differential approach for manipulating the simulation function. The animator adjusts the motion by specifying a differential change of motion $\delta\mathbf{q}_i$ in the generalized state $\mathbf{q}(t_i)$ at time t_i. The system responds by reshaping the current motion to comply with the adjustments. Continuing the interactive manipulation, the animator gradually guides the system toward the desired solution. To compute a new control vector that reshapes the motion in compliance with the differential changes $\delta\mathbf{q}_i$, we locally linearize Equation 5,

$$\delta\mathbf{q}_i = \frac{\partial \mathcal{S}(t_i, \mathbf{u})}{\partial \mathbf{u}} \delta\mathbf{u}. \tag{6}$$

We combine all animator-specified constraints into a linear system which we solve for $\delta\mathbf{u}$ by conjugate gradient technique. The differential vector $\delta\mathbf{u}$ describes the direction in which to change the current control vector $\mathbf{u}$ to obtain the desired motion change $\delta\mathbf{q}_i$. The differential update is simply a small step in the computed direction,

$$\mathbf{u}' = \mathbf{u} + \epsilon\, \delta\mathbf{u}. \tag{7}$$

Given the new, updated control vector $\mathbf{u}'$, a rigid body simulator computes the new motion and displays the result. At this point the entire process repeats.

3.2 2-D Particle Example

To help provide an intuition for the issues underlying our approach, we begin with an illustrative example. Suppose that a single 2-D particle moves under the action of gravity. The generalized state $\mathbf{q} \in \mathbf{R}^4$ encodes the particle's position $\mathbf{x} \in \mathbf{R}^2$ and velocity $\mathbf{v} \in \mathbf{R}^2$. If g is the acceleration of gravity, the equations of motion,

$$\frac{d}{dt}\begin{pmatrix}\mathbf{x}(t)\\ \mathbf{v}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{v}(t)\\ \begin{pmatrix}0\\ -g\end{pmatrix}\end{pmatrix}, \tag{8}$$

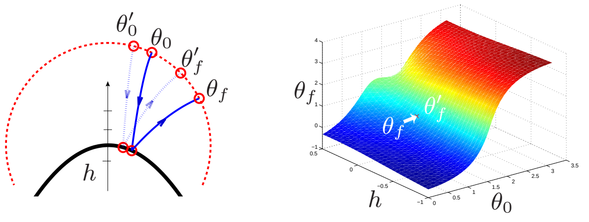


Figure 2: The simulation function for the motion of a particle bounce.

describe the particle's path in free flight. The solution to this differential equation yields the simulation function:

$$\mathcal{S}(t, \mathbf{q}_0) = \begin{pmatrix}\mathbf{x}(t)\\ \mathbf{v}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{x}(0) + \mathbf{v}(0)t + \begin{pmatrix}0\\ -\frac{1}{2}gt^2\end{pmatrix}\\ \mathbf{v}(0) + \begin{pmatrix}0\\ -gt\end{pmatrix}\end{pmatrix}. \tag{9}$$

If the particle collides with an immovable obstacle, the Poisson collision model applies an impulse to change the particle's velocity. For frictionless collisions, the impulse acts in the direction of the surface normal $\mathbf{n}$ at the point of collision. The equation,

$$\mathbf{v}^+ = \mathbf{v}^- - 2(\mathbf{n} \cdot \mathbf{v}^-)\mathbf{n}, \tag{10}$$

applies an impulse to instantaneously change the particle's velocity before the collision $\mathbf{v}^-$ into its velocity after the collision $\mathbf{v}^+$.

Given these analytical expressions for the particle's motion, we can plot the space of all possible trajectories for the particle as a function of the initial conditions and the environment. For concreteness, suppose the particle collides with a single parabolic obstacle. For notational convenience, we introduce a unit circle around the obstacle: the particle enters the circle at some angle θ_0 with unit velocity vector directed towards a point at height h above the tip, bounces off the obstacle, and exits the circle at another angle θ_f (Figure 2). Our objective is to determine θ_f as a function of θ_0 and h.

In this example, the simulation function $\mathcal{S} : \mathbf{R}^2 \to \mathbf{R}$ maps the control vector $\mathbf{u} = (\theta_0, h)$ into the particle's final, exit position θ_f. Given an initial entering θ_0 and exiting θ_f state, our gradient-based interactive technique can smoothly transform this solution to one which satisfies one or more constraints, for example to achieve a different exiting state θ_f'. Our technique converges easily because the simulation function is smooth over the domain of control parameters (Figure 2).

The general motion of many rigid bodies is much like this simple particle example. To describe the state of a single 3-D rigid body, we increase the dimensions of the generalized state, adding the components of orientation, angular velocity, and extending the position and linear velocity to 3-D. Two or more rigid bodies are modeled by adding additional components to the generalized state. Surface parameters such as normals and elasticity coefficients may also be added, if desired. Note that the number of rigid objects is not explicitly represented, we are merely expressing the cumulative degrees of freedom of the system. Our implementation makes use of this representation to enable complex multi-object simulations with the same computation techniques and data structures used to implement particle simulations.

4 Manipulation without Discontinuities

The algorithm outlined in Section 3 relies on the efficient computation of the Jacobian matrix $\partial\mathcal{S}(t_i, \mathbf{u})/\partial\mathbf{u}$. Computing the Jacobian matrix with finite differences is expensive because of the need to perform multiple simulations. In addition, the inaccuracies of the finite differences approach would be prohibitive for our approach.

Instead we use a specialized automatic differentiation technique. We decompose the simulation function $\mathcal{S}$ into analytically differentiable functions and numerically compose the Jacobian matrix using the chain rule. For example, suppose that a single collision occurs at time t_c and the simulation function $\mathcal{S}(t_f, \mathbf{u})$ describes the body's state at some time after the collision $t_f > t_c$. We decompose $\mathcal{S}(t_f, \mathbf{u})$ into three functions:

$\mathcal{F}_{t_c}$**:** pre-collision free-flight function, which maps the initial conditions and perhaps additional elements of the control vector $\mathbf{u}$ into the body's state at t_c, an instant before collision (e.g. Equation 9 for 2-D particles);

$\mathcal{C}_{t_c}$**:** collision function, which applies the impulse and maps the body's state an instant before collision into the body's state at t_c, an instant after collision (e.g. Equation 10 for 2-D particles);

$\mathcal{F}_{t_f}$**:** post-collision free-flight function, which maps the body's state an instant after the collision into the body's state at t_f.

The functional composition expressing $\mathcal{S}(t_f, \mathbf{u})$ becomes:[1]

$$\mathcal{S}(t_f, \mathbf{u}) = \mathcal{F}_{t_f}(\mathbf{u}) \circ \mathcal{C}_{t_c}(\mathbf{u}) \circ \mathcal{F}_{t_c}(\mathbf{u}). \tag{11}$$

Although the free-flight motion of the particle in Section 3.2 has a closed-form and is analytically differentiable, this is generally not the case for 3-D rigid body motion.[2] To compute the derivatives of $\partial\mathcal{F}_{t_c}(\mathbf{u})/\partial\mathbf{u}$, we first integrate the equations of motion (Equation 1) until time t_c,

$$\mathcal{F}_{t_c}(\mathbf{u}) = \mathbf{q}_0(\mathbf{u}) + \int_{t_0}^{t_c(\mathbf{u})} \mathbf{F}(t, \mathbf{q}, \mathbf{u})\, dt,$$

and take the derivative of both sides with respect to $\mathbf{u}$

$$\frac{\partial\mathcal{F}_{t_c}(\mathbf{u})}{\partial\mathbf{u}} = \frac{\partial}{\partial\mathbf{u}}\left(\mathbf{q}_0(\mathbf{u}) + \int_{t_0}^{t_c(\mathbf{u})} \mathbf{F}(t, \mathbf{q}, \mathbf{u})\, dt\right).$$

To evaluate this expression we apply the Leibnitz rule [18] to interchange the integral and the derivative:[3]

$$\begin{aligned}\frac{\partial\mathcal{F}_{t_c}(\mathbf{u})}{\partial\mathbf{u}} = {} & \mathbf{F}(t_c(\mathbf{u}), \mathbf{q}, \mathbf{u})\frac{dt_c(\mathbf{u})}{d\mathbf{u}} + \\ & \frac{\partial\mathbf{q}_0(\mathbf{u})}{\partial\mathbf{u}} + \int_{t_0}^{t_c(\mathbf{u})} \frac{\partial\mathbf{F}(t, \mathbf{q}, \mathbf{u})}{\partial\mathbf{u}}\, dt.\end{aligned} \tag{12}$$

The simulator computes the value of $\mathbf{F}(t_c(\mathbf{u}), \mathbf{q}, \mathbf{u})$ at the collision. To compute the collision time derivative $dt_c(\mathbf{u})/d\mathbf{u}$ we define a smooth collision event function $E(t, \mathbf{q})$ such that at the collision time $t_c(\mathbf{u})$,

$$E(t_c(\mathbf{u}), \mathbf{q}) = 0. \tag{13}$$

For the 2-D particle, for example, the collision event function E can be defined as the signed distance function between the particle and the obstacle.

[1]The Equation 11 is written in this form for notational convenience. More precisely, this equation is $\mathcal{S}(t_f, \mathbf{u}) = \mathcal{F}_{t_f}(\mathbf{u}, \mathcal{C}_{t_c}(\mathbf{u}, \mathcal{F}_{t_c}(\mathbf{u})))$.

[2]For the special case of freely rotating 3-D rigid body (no torques), there is an analytic Poinsot's solution [26].

[3]The conditions for applying the Leibnitz rule require that $\mathbf{F}$ is continuous and has a continuous derivative $\partial\mathbf{F}/\partial\mathbf{u}$. These conditions are met under reasonable assumptions about external forces.

Differentiating Equation 13 and solving for the collision time derivative we obtain

$$\frac{dt_c(\mathbf{u})}{d\mathbf{u}} = -\frac{(\partial E/\partial\mathbf{q}) \cdot (\partial\mathbf{q}/\partial\mathbf{u})}{\partial E/\partial t}. \tag{14}$$

The derivatives on the right-hand side of Equation 14 are computed analytically, with the exception of $\partial\mathbf{q}/\partial\mathbf{u}$, which is defined by the integral expression (second and third term in the sum) in Equation 12. We compute this integral expression by numerically integrating differential equation

$$\frac{d}{dt}\frac{\partial\mathbf{q}(t)}{\partial\mathbf{u}} = \frac{\partial\mathbf{F}(t, \mathbf{q}, \mathbf{u})}{\partial\mathbf{u}},$$

until time t_c with the initial condition $\partial\mathbf{q}_0(\mathbf{u})/\partial\mathbf{u}$.

The computation of $\partial\mathcal{F}_{t_f}(\mathbf{u})/\partial\mathbf{u}$ is similar: we apply the Leibnitz rule to obtain

$$\begin{aligned}\frac{\partial\mathcal{F}_{t_f}(\mathbf{u})}{\partial\mathbf{u}} = {} & -\mathbf{F}(t_c(\mathbf{u}), \mathbf{q}, \mathbf{u})\frac{dt_c(\mathbf{u})}{d\mathbf{u}} + \\ & \frac{\partial\mathcal{C}_{t_c}(\mathbf{u})}{\partial\mathbf{u}} + \int_{t_c(\mathbf{u})}^{t_f} \frac{\partial\mathbf{F}(t, \mathbf{q}, \mathbf{u})}{\partial\mathbf{u}}\, dt\end{aligned}$$

and evaluate the right-hand terms as before.

To compute the derivatives of $\partial\mathcal{C}_{t_c}(\mathbf{u})/\partial\mathbf{u}$ we differentiate the Equation 4:

$$\frac{\partial\mathcal{C}_{t_c}(\mathbf{u})}{\partial\mathbf{u}} = \frac{\partial\mathcal{F}_{t_c}(\mathbf{u})}{\partial\mathbf{u}} + \frac{\partial\mathbf{I}(\mathbf{q}^-, \mathbf{u})}{\partial\mathbf{u}}.$$

Once all derivatives of the sub-functions have been computed we find the simulation function derivatives by applying the chain rule:

$$\frac{\partial\mathcal{S}(t_f, \mathbf{u})}{\partial\mathbf{u}} = \frac{\partial\mathcal{F}_{t_f}}{\partial\mathcal{C}_{t_c}}\left(\frac{\partial\mathcal{C}_{t_c}}{\partial\mathcal{F}_{t_c}}\frac{\partial\mathcal{F}_{t_c}}{\partial\mathbf{u}} + \frac{\partial\mathcal{C}_{t_c}}{\partial\mathbf{u}}\right) + \frac{\partial\mathcal{F}_{t_f}}{\partial\mathbf{u}}$$

Although we have shown the derivative computations for the composition of three phases, an arbitrary number of such phases can be composed in an analogous manner.

4.1 Differential Update

Having computed the Jacobians, we can formulate the constraint equations (Equation 6). Given n such equations, we solve for the differential vector $\delta\mathbf{u}$. Because this system is often underconstrained (Section 1), we solve the following minimization instead:

$$\begin{aligned} & \min_{\delta\mathbf{u}} \; \left(\delta\mathbf{u}^T\mathbf{M}\,\delta\mathbf{u} + \mathbf{d}^T\delta\mathbf{u}\right) \\ \text{subject to} \quad & \delta\mathbf{q}_1 = \frac{\partial\mathcal{S}(t_1, \mathbf{u})}{\partial\mathbf{u}}\delta\mathbf{u} \\ & \vdots \\ & \delta\mathbf{q}_n = \frac{\partial\mathcal{S}(t_n, \mathbf{u})}{\partial\mathbf{u}}\delta\mathbf{u}. \end{aligned} \tag{15}$$

The minimized objective function has a dual purpose: it seeks the smallest change from the current state of the simulation and the smallest deviation from the desired values of the simulation parameters such as surface normals at the collision. The diagonal matrix $\mathbf{M}$ describes the relative scale between parameters in the control vector $\mathbf{u}$. The animator can describe the desired scaling to specify how the system should change the parameters. For example, the animator may instruct the system to favor changing the initial position rather than the initial velocity of a body. The vector $\mathbf{d}$ defines desired values for physical parameters. For example, if the system varies the surface normal at a collision we can specify the true geometric normal as the desired value and the system will attempt to stay as close as possible—once all constraints are satisfied—to the true surface normal. Specifically, if $\delta\mathbf{u}_d$ is the desired change in the control vector $\mathbf{u}$ then setting $\mathbf{d} = -\delta\mathbf{u}_d$ and optimizing Equation 15 will minimize $(\delta\mathbf{u} - \delta\mathbf{u}_d)^T(\delta\mathbf{u} - \delta\mathbf{u}_d)$. Because the objective is quadratic and all constraints are linear, we use the Lagrangian multipliers to reformulate the minimization as a linear system and solve for $\delta\mathbf{u}$ [12].

Our technique is a form of gradient descent: we continuously linearize the problem and move in the gradient direction $\delta\mathbf{u}$. For a large gradient stepsize ϵ, the gradient descent method may diverge. Line minimization is the preferred method for choosing the stepsize in a gradient method, but it requires considerable computation. In practice, a small fixed stepsize has good convergence properties while also enabling interactive update rates.

The gradient descent converges only to a local optimum [5]. Local convergence is sufficient and effective for our interactive setting: the animator drags a body towards the intended position—guiding the system out of undesirable local minima—and the system quickly reshapes the motion to comply with the change.

4.2 Manipulation Constraints

When the animator specifies the constraints, the system maps these constraints to the appropriate differential changes of motion $\delta\mathbf{q}_i$. We distinguish three types of constraints: state constraints, expression constraints, and floating constraints.

State constraints occur when the animator "nails down" objects (e.g., fixing position, orientation, linear velocity or angular velocity to specific values). Suppose that the animator wants the body A at time t_i to have the state $\mathbf{q}'_A$, and that $\mathbf{q}_A$ is a subset of the generalized state of the whole system $\mathbf{q}$ which describes the state of the body A. We write the desired differential change as $\mathbf{q}'_A - \mathbf{q}_A(t_i)$. In this case the nail constraint is enforced at a specific time instant t_i.

Expression constraints are generalizations of the state constraints. Any differentiable expression of the generalized state $\mathbf{q}$ can represent a constraint. For example, the animator can equate the speed of two bodies with the constraint $|\mathbf{v}(\mathbf{q}_A(t_i))| - |\mathbf{v}(\mathbf{q}_B(t_i))|$.

Both state and expression constraints can be specified without fixing the time of evaluation t_i. The animator can express a constraint at a particular event—say, the fifth collision in the simulation. Time of collision $t_c(\mathbf{u})$ is not fixed and thus the time of the constraint can "float." For example, we can reduce the angular velocity ω of body A with the constraint $-\omega(\mathbf{q}_A(t_i)) \cdot \omega(\mathbf{q}_A(t_i))$. Subsequent modification of various simulation parameters will change the time at which the collision occurs, but the floating constraint will still be enforced.

5 Manipulation with Discontinuities

When the simulation function is continuous, the interactive manipulation technique described in Section 4 effectively converges to the desired motion. In general, however, the simulation function contains discontinuities that may cause this technique to diverge. In this section we describe a method for improving the convergence for piecewise continuous simulation functions.

The simulation function is discontinuous whenever polygonal (piecewise linear) meshes are involved in collisions. For example, suppose we modify the particle example from Section 4 and replace the smooth, curved obstacle with a piecewise linear polygonal curve (Figure 3). As long as the particle collides with the same edge, the simulation function remains continuous. On the other hand, when the particle collides with a different edge, the surface normal on the

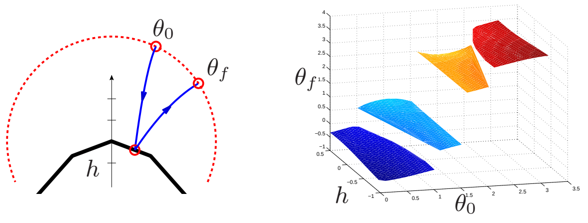


Figure 3: Sample particle bounce motions with polygonal obstacle and the corresponding piecewise smooth simulation function

obstacle changes abruptly and thus the collision impulse applied in Equation 10 is discontinuous. This abrupt change carries over to the subsequent particle motion and corresponds to a discontinuity in the simulation function. We cannot disregard piecewise linear approximations because the interactive rigid body simulators often approximate smooth geometric models with polygonal (piecewise linear) meshes—mostly because meshes facilitate faster and easier collision detection.

In general, the simulation function is piecewise continuous. A connected set of control vectors for which the simulation function is continuous defines a connected component in the control space. We call these connected components *smooth components* because on a smooth component the simulation function is continuously differentiable. For example, a set of control vectors for which the particle collides with the same edge of the obstacle defines a smooth component (Figure 3). In this example, the four smooth components correspond to motions of the particle colliding with each of the four edges. The figure emphasizes two main problems caused by discontinuities: the loss of physical feasibility and degradation of convergence. We describe these problems and our solutions in the remainder of this section.

5.1 Physical Feasibility

As shown in Figure 3, the polygonal approximation of the obstacle restricts the physically feasible exit points for the particle. Note that some values of θ_f are unattainable because the surface normal near the origin is discontinuous: the particle cannot exit at the section of the circle directly above the origin (θ_f near $\pi/2$). This restriction of feasible results becomes especially evident when the animator over-constrains the system with many desired body configurations. Finer polygonal approximations reduce the gaps in the piecewise smooth function, but overly fine approximations increase the collision detection time and reduce interactivity.

Our approach to this problem is twofold. First, we introduce additional control parameters to vary the surface normals on a polygonal mesh and to simulate a collision with a smooth obstacle. If the mesh approximates a smooth surface the desired normal can be computed from a smooth local interpolant or, if available, from the true smooth surface. The normal can then be adjusted dynamically by including the normal deviation within the control vector $\mathbf{u}$. As Figure 4 illustrates varying surface normals extends the range of smooth components to increase the physically feasible regions.

Second, we use curvature-dependent polygonal approximations in our simulations because they keep the facet count low for fast collision detection and simulation, but also provide good first-order approximations to the original surface [11]. For discontinuities due to polygonal approximations of smooth surfaces, the computed differential change $\delta\mathbf{u}$ continues to contain valuable information. Approximating smooth surfaces with polygonal meshes is well studied in computer graphics. In general, good approximations allocate many facets to areas of high surface curvature and fewer facets to near-planar surface regions. For these polygonal meshes, despite the discontinuity in the surface normals, the currently collid-

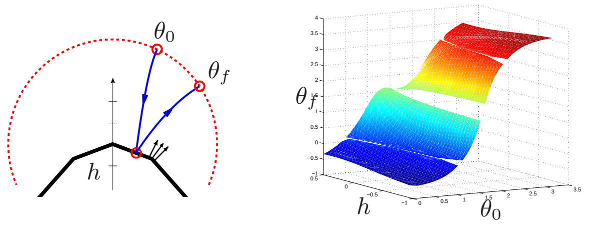


Figure 4: Varying surface normals reduces the gaps to increase physically feasible regions.

ing facet is also a good first-order approximation to the underlying surface. In this case the differential change $\delta\mathbf{u}$ continues to be a good predictor for the differential update because the first-order approximation is sufficiently accurate for linear Equation 6.

5.2 Convergence

The interactive technique of Section 4 converges to the desired motion if there exists a path from the initial to the desired control vector within a single smooth component. With discontinuities, such a path may not exist. The discrete search must guide the control vector between the appropriate components, piecing together a path that crosses discontinuities. Especially in higher dimensions, this is a daunting task for an interactive system. In general, the search must take into account physically feasible regions and jump to smooth components in possibly distant regions of a high-dimensional control space. The most important criterion for selecting smooth components is that they facilitate convergence to the desired motion. In addition, unless instructed otherwise, the components should preserve the "style" of the current motion, as that may be of primary importance. For example, if an animator desires a successful "off-the-backboard" basketball shot, it is undesirable to jump to a smooth component corresponding to a direct, "nothing-but-net" motion. Lastly, the discrete search must complete quickly to maintain interactivity. Our solution relies on two concepts: sampling and interaction.

Sampling In the presence of discontinuities our technique becomes more sensitive to the stepsize ϵ and the direction $\delta\mathbf{u}$ in the differential update (Equation 7). With a large stepsize ϵ, the gradient-descent method may diverge. The approximation errors in $\delta\mathbf{u}$ also adversely affect convergence. To improve convergence, we use sampling to find the best values for these parameters. To find a good stepsize ϵ we use a form of the successive stepsize reduction technique:[4] our discrete search chooses an initial stepsize ϵ and reduces it a few times to select the motion that most closely matches the desired result. Convergence results for gradient methods with non-random errors, such as approximation errors in $\delta\mathbf{u}$, exist [5], but there are no standard techniques for improving the convergence. Recall from Section 5.1 that for discontinuities due to polygonal approximations, the update vector $\delta\mathbf{u}$ is a good heuristic for the new samples. Thus, when the simulation is directed off the edge of the smooth component, our system samples the control space from the normal distribution centered around the suggested update $\delta\mathbf{u}$. Each such sample may produce a point on a new smooth component. We evaluate how well the corresponding motions comply with the constraints and jump to the most promising component. The animator perceives the jump as a minor "pop" in the resulting motion and typically, following the jump, the continuous manipulation continues. The sampling procedure also causes a momentary lag. While the lag could be reduced with a faster implementation, the visual pop is unavoidable in situations where the underlying motion is discontinuous. If sampling does not produce any reasonable smooth component, the system remains within the current smooth component. The animator is thus blocked from adjusting the motion in a particular way, but can continue to guide the system in a different way.

[4]Successive stepsize reduction is not theoretically sound because the improvement at each iteration is not enough to guarantee convergence. Nevertheless, it often works in practice [5].

Interaction Of course, to guarantee convergence we would have to search through the entire control space. Our system does not address this more general problem—the high dimension of the control space makes the search especially difficult. Instead, our technique relies on the animator to guide the system to a motion that satisfies given constraints. For example, a body that initially flies over a wall may have to bounce off the wall and fly in the opposite direction to accomplish the desired constraint. Our technique will not make these transformations automatically. For a large class of motion design tasks, this behavior is desirable and sufficient. The interaction allows the animator to quickly experiment and guide the system toward the desired collision sequence. For example, to transform the motion of a basketball during a successful free throw, the animator may want to bounce the ball off the backboard before it goes through the hoop. In this case, the animator first guides the ball into a backboard collision, and then guides it through the hoop. We emphasize that the single constraint specifying a successful shot does not uniquely determine the desired collision configurations: the ball may bounce off the backboard, off the floor or even off the scoreboard. An automatic system would have to choose the desired motion (or keep track of a possibly exponential number of motions) according to some objective criteria. Instead, our system provides the animator with interactive, direct control over the motion and allows her to guide the system to the appropriate solution.

6 Implementation and Results

Implementation The implementation of our system is decomposed into three parts: (1) a differential control module, (2) a rigid body dynamics simulator, and (3) a user interface for motion display and editing. The control of the system is animator-driven. In response to an edit (a mouse event), the control module recomputes the control parameters $\mathbf{u}$ needed to accomplish the desired motion adjustments. These parameters are then provided to the physical simulator, which recomputes the motion and updates the display.

We use the general-purpose rigid body simulator developed by Baraff [1]. Alternatively, specialized simulators could be used that provide tighter integration with the differential control module. Our manipulation tool controls the simulator at two points: (1) it provides the control vector $\mathbf{u}$ for the simulation and (2) it modifies the impulses at collisions using the modified surface normals and elasticity coefficients. The simulator, in turn, computes the new motion and returns the new collision events. The computed motion is used to update the display and the collisions are used to define a new expression for the equations of motion (Equation 11).

For example, a single-bounce motion has a decomposition corresponding to Equation 11. A change in the control parameters may cause another bounce to occur. In this case, the simulator detects the additional collision. In response, our system automatically updates the equations of motion by adding an additional collision function and two more flight phases to expression in Equation 11.

We use the exponential map parameterization of orientations [15] in the control vector $\mathbf{u}$, finding that it yields better results than the normalized quaternions.

Examples This section demonstrates the use of our system to construct several physically based animations. All of these examples were created by direct manipulation in real-time, and each required between two and ten minutes of editing to achieve the desired animation. For each of these examples, Figure 5 shows the animations before interaction, at an intermediate point, and after the desired motion is obtained. Each image in the figure displays the entire simulation by tracing out the trajectories of one or two points on the moving objects (shown in black). After experimenting with a variety of different interfaces, we have found that this display minimizes clutter yet provides the animator with a sense of the cumulative motion that is sufficient for most interaction tasks. Of course the animator can choose to view the complete motion as a traditional frame sequence at any time during the interaction.

The objective of the first example is to have two eggs collide in the air and land successfully into two buckets on the ground (Figure 5(a)). Creating such a motion by simply adjusting initial positions and velocities of the objects would be extremely difficult due to the complexity of the motion and the constraint that the buckets themselves cannot be moved. In contrast, the desired animation is easily created from scratch using our interactive manipulation technique. First, the starting positions of the eggs are fixed, and the velocities and orientations are assigned arbitrarily. By clicking at a point on its flight path, the animator then interactively drags the first egg's trajectory towards the second egg so that the two objects collide in the air. Running at roughly 20 frames per second, the system computes the required changes in the initial orientation and velocity of *both* eggs to achieve the desired motion updates. Once one egg is in the bucket, the animator applies a nail constraint to fix its ending state and then drags the second egg into the other bucket.

In the second example, the animator's goal is to drop a plank onto two supports to form a table. The problem is made more difficult by requiring the plank to collide with a pyramid object in the air, prior to landing centered on the supports. This example requires the system to solve for the initial plank position, orientation, and velocity (both linear and angular) in order to achieve the desired configuration after the collision. Like the previous example, this is achieved by allowing the animator to directly manipulate the plank's desired position and orientation while the system interactively computes the corresponding physical parameters. This manipulation occurs in two steps: first the animator selects the plank after it collides with the pyramid, and positions it above the supports. Second, the plank's orientation is interactively aligned so that it lands squarely on the supports (Figure 5(b)).

The third example demonstrates the use of normal and elasticity parameters to aid editing operations, and the use of floating time constraints. Suppose the animator wishes to keep a falling mug from tipping over without changing its initial position, orientation, or velocity. This is accomplished by adding new control parameters to control the surface normal and elasticity parameters of the floor at the points where the mug hits the floor. To keep the mug from tipping over, the animator first straightens the mug so that it is upright at the fourth bounce. The system accommodates this change by modifying the floor normal at the third bounce. Note that this change in the normal will typically alter the time at which the fourth bounce occurs, requiring a floating time constraint (Section 4.2).

Due to its angular velocity, however, the mug still tips over (Figure 5(c), center). This is prevented by constraining its angular velocity to be zero after the fourth bounce, resulting in a motion where the mug does not tip over (right). The changes in surface normals are perceived as changes in the surface texture of the floor.

The final example illustrates the ability to edit the style of an animation by modifying a previously constructed motion. In this example, a scissors is thrown into the air and lands on a coatrack (Figure 5(d)). This initial animation is constructed by starting with a motion in which the scissors falls on the floor and then interactively dragging it to the coatrack. By selecting and manipulating the scissors at different points in its trajectory, this motion is transformed into one in which the scissors first bounces off the ground, performs a somersault in the air, and still successfully lands on the coatrack. This example demonstrates how progressively more interesting and complex motions may be created from simpler motions using our interactive editing approach.

7 Conclusion

In this paper, we have described a new interactive technique for manipulating rigid multi-body simulations. Instead of changing the simulation parameters directly, we provide an intuitive interface through which the animator can manipulate the position and velocity of objects directly. Using our system, the animator can rapidly design difficult physical animations that would be difficult to achieve with existing rigid body simulators.

For some design tasks, the interactive paradigm is not the most effective. For example, the animator may be hard pressed to chart out the sequence of collisions that will lead *all* billiard balls into pockets. In general, this is a difficult problem that, in some cases, may not even have a solution. Other motion-construction techniques [7, 27] address these problems and, in some scenarios, construct appropriate motions after extensive computation. We envision a hybrid system that integrates a motion-construction technique with our interactive manipulation tool to improve the effectiveness of the interactive paradigm.

For the Jacobian evaluation (Section 4), our technique assumes that the collision function is analytically differentiable. This is not always the case with the rigid body simulator we use in our prototype implementation. During a resting (i.e. sustained) contact or for multiple simultaneous collisions the applied impulses are solutions to a linear complementarity problem (LCP) [1]. In general, LCP problems do not have closed-form, analytically differentiable solutions. There are many alternative formulations which may facilitate analytic differentiation. Further, the interactive manipulation technique would benefit from a specialized rigid body simulator. For example, the simulator could simultaneously integrate both body states and their derivatives.

Lastly, the interactive manipulation is not possible for all rigid multi-body systems: in some scenarios simulation alone requires considerable computational time. In these cases the animators will have to resort to an off-line motion-construction technique.

Acknowledgements

We would like to thank the members of the CMU graphics lab for helping us shape our ideas. Sebastian Grassia and Paul Heckbert provided valuable suggestions and insights. We especially thank David Baraff for kindly providing the rigid body simulator. We also thank Elly Winner and Ivan Sokić for helping us in the final stages of our paper submission. The support of the Microsoft Corporation is gratefully acknowledged.

References

[1] David Baraff. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *Computer Graphics (Proceedings of SIGGRAPH 94)*, Annual Conference Series, pages 23–34. ACM SIGGRAPH, July 1994.

[2] David Baraff and Andrew Witkin. Large Steps in Cloth Simulation. In *Computer Graphics (Proceedings of SIGGRAPH 98)*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, July 1998.

[3] Ronen Barzel and Alan H. Barr. A Modeling System Based On Dynamic Constraints. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, Annual Conference Series, pages 179–188. ACM SIGGRAPH, August 1988.

[4] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible Motion Simulation for Computer Graphics Animation. In *Computer Animation and Simulation '96*, Proceedings of the Eurographics Workshop, pages 184–197, Poitiers, France, September 1996.

[5] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 1995.

[6] Lynne Shapiro Brotman and Arun N. Netravali. Motion Interpolation by Optimal Control. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 26 of *Annual Conference Series*, pages 309–315. ACM SIGGRAPH, August 1988.

[7] Stephen Chenney and D. A. Forsyth. Sampling Plausible Solutions to Multi-body Constraint Problems. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*, Annual Conference Series. ACM SIGGRAPH, July 2000.

[8] Michael F. Cohen. Interactive Spacetime Control for Animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, Annual Conference Series, pages 293–302. ACM SIGGRAPH, July 1992.

[9] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic Motion Planning. *Journal of the ACM*, 40(5):1048–1066, November 1993.

[10] Nick Foster and Dimitri Metaxas. Realistic Animation of Liquids. *Graphical Models and Image Processing*, 5(58):471–483, 1996.

[11] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Computer Graphics (Proceedings of SIGGRAPH 97)*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, August 1997.

[12] Philip E Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, London, 1989.

[13] Michael Gleicher and Andrew Witkin. Differential Manipulation. In *Graphics Interface*, pages 61–67, June 1991.

[14] Michael Gleicher and Andrew Witkin. Through-the-Lens Camera Control. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, Annual Conference Series, pages 331–340. ACM SIGGRAPH, July 1992.

[15] F. Sebastian Grassia. Practical Parameterization of Rotation Using the Exponential Map. *Journal of Graphics Tools*, 3(3):29–48, 1998.

[16] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. In *Computer Graphics (Proceedings of SIGGRAPH 98)*, Annual Conference Series, pages 9–20. ACM SIGGRAPH, July 1998.

[17] Mikako Harada, Andrew Witkin, and David Baraff. Interactive Physically-Based Manipulation of Discrete/Continuous Models. In *Computer Graphics (Proceedings of SIGGRAPH 95)*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, August 1995.

[18] Wilfred Kaplan. *Advanced Calculus*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[19] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical Spacetime Control. In *Computer Graphics (Proceedings of SIGGRAPH 94)*, Annual Conference Series, pages 35–42. ACM SIGGRAPH, July 1994.

[20] Matthew Moore and Jane Wilhelms. Collision Detection and Response for Computer Animation. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, Annual Conference Series, pages 289–298. ACM SIGGRAPH, August 1988.

[21] J. Thomas Ngo and Joe Marks. Spacetime Constraints Revisited. In *Computer Graphics (Proceedings of SIGGRAPH 93)*, Annual Conference Series, pages 343–350. ACM SIGGRAPH, August 1993.

[22] James F. O'Brien and Jessica K. Hodgins. Graphical Modeling and Animation of Brittle Fracture. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, Annual Conference Series, pages 111–120. ACM SIGGRAPH, August 1999.

[23] Zoran Popović and Andrew Witkin. Physically Based Motion Transformation. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, August 1999.

[24] Jos Stam. Stable Fluids. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, Annual Conference Series, pages 121–128. ACM SIGGRAPH, August 1999.

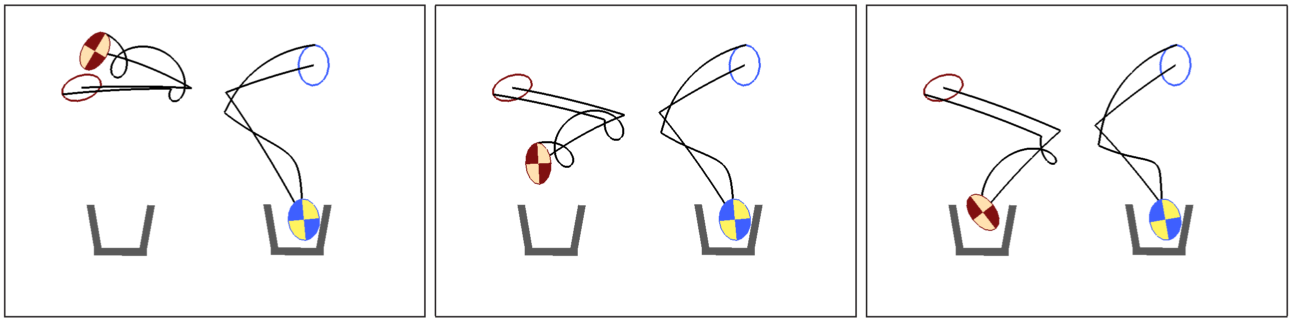
[25] Robert F. Stengel. *Optimal Control and Estimation*. Dover Books on Advanced Mathematics, New York, 1994.

[26] Keith R. Symon. *Mechanics, Third Edition*. Addison-Wesley Publishing Company, Reading, Massachussetts, 1971.

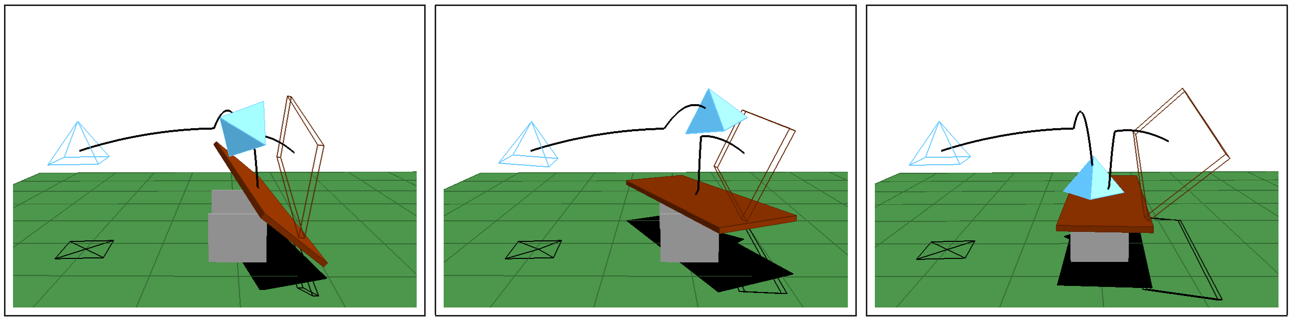
[27] Diane Tang, J. Thomas Ngo, and Joe Marks. N-Body Spacetime Constraints. *Journal of Visualization and Computer Animation*, 6:143–154, 1995.

[28] Andrew Witkin, Michael Gleicher, and William Welch. Interactive Dynamics. In *Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 11–21, March 1990.

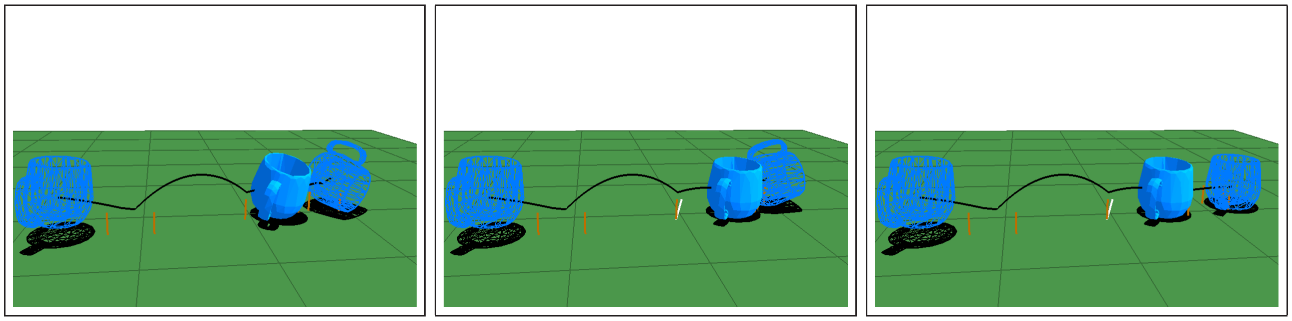
[29] Andrew Witkin and Michael Kass. Spacetime Constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, Annual Conference Series, pages 159–168. ACM SIGGRAPH, August 1988.



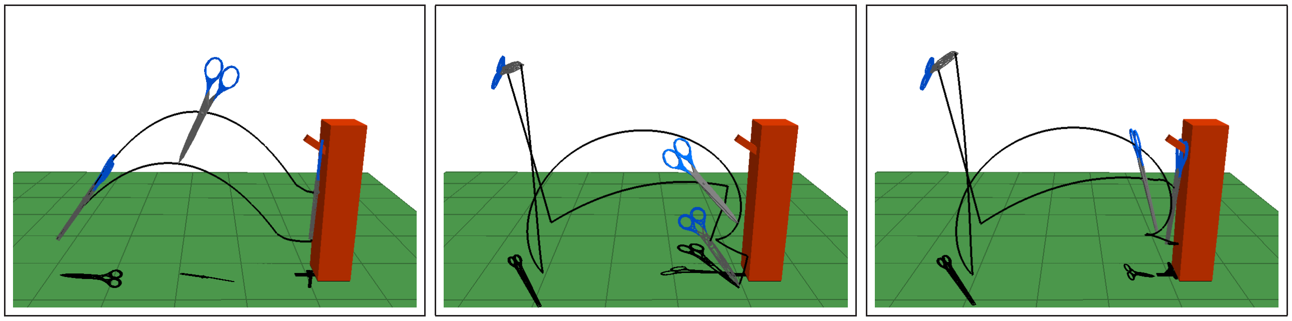
(a)



(b)



(c)



(d)

Figure 5: Physical motions (left) are interactively edited to satisfy desired constraints (right). Intermediate motions during the editing process are shown at center. (a) An egg is dragged into a bucket after collision with a second egg. The second egg is required to fall into a second bucket with a nail constraint. (b) A table top is made to land on its legs after collision with a pyramid. (c) A tumbling mug is kept from tipping over by editing its orientation and angular velocity at the fourth collision with the ground. (d) A bounce and a flip is added to an animation where a scissors lands on a coatrack. All interaction occurs in real time.